CZ1104 - Lab 2 (Revised)

Submitted by Dyllon

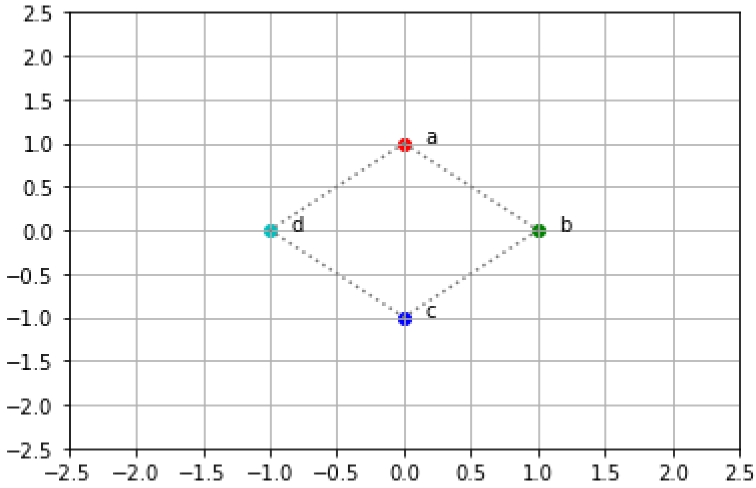
In [1]:
```
# essential imports - from the supploied docx
import matplotlib.pyplot as plt
import numpy as np
import string
```

Exercise 1 - Computer Graphics – Linear Transformations

Question 1

Use the code available in NTULearn to plot these points. Note the application of Identity transformation in the code.

In [2]:
```
# points a, b and, c
a, b, c, d = (0, 1, 0), (1, 0, 1), (0, -1, 2), (-1, 0, 3)

# matrix with row vectors of points
A = np.array([a, b, c, d])

# 3x3 Identity transformation matrix
I = np.eye(3) #float

color_lut = 'rgbc' #4 colors to represent 4 points

fig = plt.figure()
ax = plt.gca()
xs = []
ys = []
for row in A:
    output_row = I @ row
    x, y, i = output_row
    xs.append(x)
    ys.append(y)
    i = int(i) # convert float to int for indexing
    c = color_lut[i]
    plt.scatter(x, y, color=c)
    plt.text(x + 0.15, y, f"{string.ascii_letters[i]}")
xs.append(xs[0])
ys.append(ys[0])
plt.plot(xs, ys, color="gray", linestyle='dotted')
ax.set_xticks(np.arange(-2.5, 3, 0.5))
ax.set_yticks(np.arange(-2.5, 3, 0.5))
plt.grid()
plt.show()
```



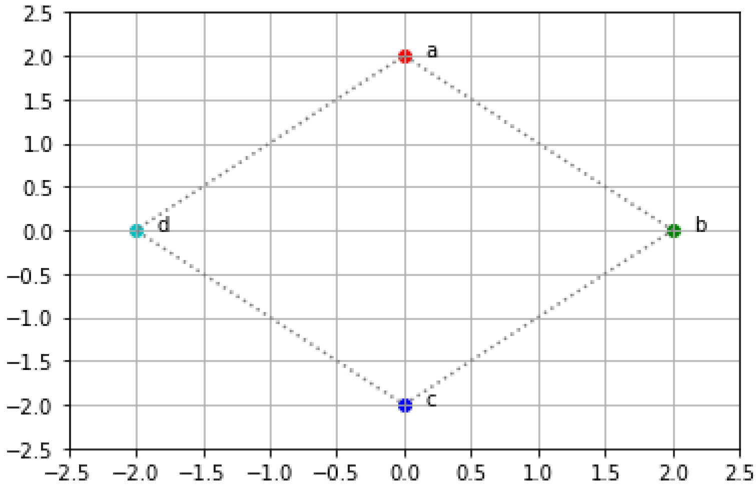
Question 2

Modify the above code to implement and the display the results of the following transformations: (i) scaling transformation with scale of 2, (ii) rotation transformation with 90°, (iii) translation, horizontal shear and vertical shear using your own parameters.

(i) Scaling transformation of scale 2

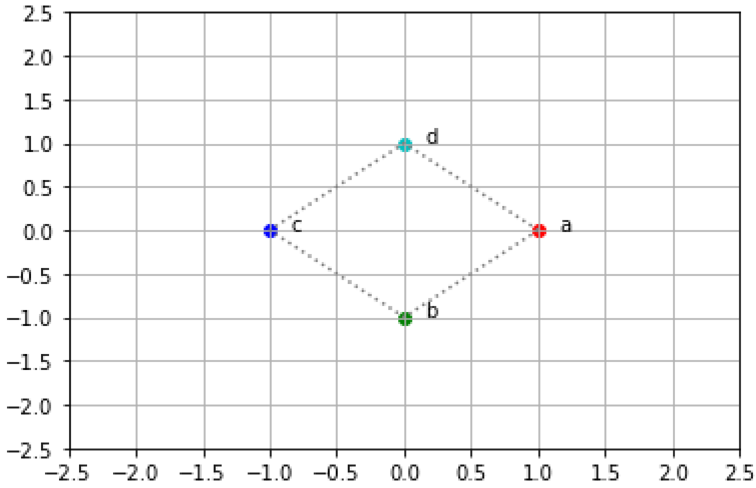
In [3]:

```
# points a, b and, c
a, b, c, d = (0, 1, 0), (1, 0, 1), (0, -1, 2), (-1, 0, 3)

# matrix with row vectors of points
A = np.array([a, b, c, d])

# Scalar transformation of scale 2
S = [
    [2, 0, 0],
    [0, 2, 0],
    [0, 0, 1]
]

color_lut = 'rgbc' #4 colors to represent 4 points

fig = plt.figure()
ax = plt.gca()
xs = []
ys = []
for row in A:
    output_row = S @ row  # @ is matrix multiplication
    x, y, i = output_row
    xs.append(x)
    ys.append(y)
    i = int(i) # convert float to int for indexing
    c = color_lut[i]
    plt.scatter(x, y, color=c)
    plt.text(x + 0.15, y, f"{string.ascii_letters[i]}")
xs.append(xs[0])
ys.append(ys[0])
plt.plot(xs, ys, color="gray", linestyle='dotted')
ax.set_xticks(np.arange(-2.5, 3, 0.5))
ax.set_yticks(np.arange(-2.5, 3, 0.5))
plt.grid()
plt.show()
```



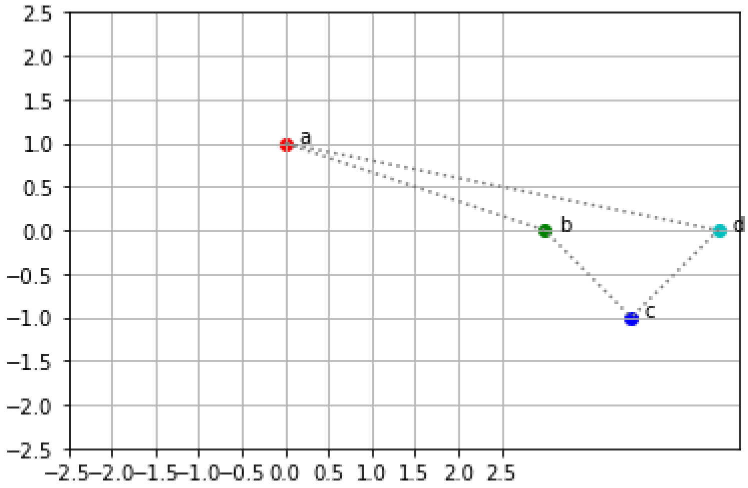
(ii) rotation transformation with 90°

In [4]:
```
# imports for trig
from math import cos, sin, radians

# points a, b and, c
a, b, c, d = (0, 1, 0), (1, 0, 1), (0, -1, 2), (-1, 0, 3)

# matrix with row vectors of points
A = np.array([a, b, c, d])

# Rotation of 90deg
R = [
    [cos(radians(90)), sin(radians(90)), 0],
    [-sin(radians(90)), cos(radians(90)), 0],
    [0, 0, 1]
]

color_lut = 'rgbc' #4 colors to represent 4 points

fig = plt.figure()
ax = plt.gca()
xs = []
ys = []
for row in A:
    output_row = R @ row  # @ is matrix multiplication
    x, y, i = output_row
    xs.append(x)
    ys.append(y)
    i = int(i) # convert float to int for indexing
    c = color_lut[i]
    plt.scatter(x, y, color=c)
    plt.text(x + 0.15, y, f"{string.ascii_letters[i]}")
xs.append(xs[0])
ys.append(ys[0])
plt.plot(xs, ys, color="gray", linestyle='dotted')
ax.set_xticks(np.arange(-2.5, 3, 0.5))
ax.set_yticks(np.arange(-2.5, 3, 0.5))
plt.grid()
plt.show()
```



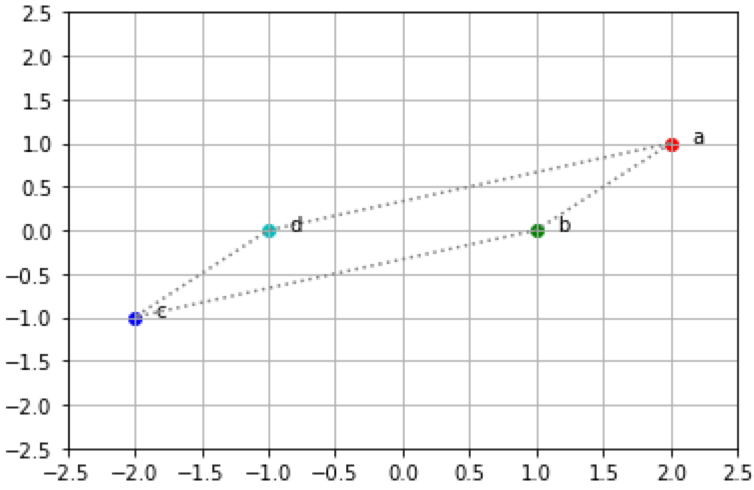
(iii) translation, horizontal shear and vertical shear

In [5]:
```
# imports for trig
from math import cos, sin, radians

# points a, b and, c
a, b, c, d = (0, 1, 0), (1, 0, 1), (0, -1, 2), (-1, 0, 3)

# matrix with row vectors of points
A = np.array([a, b, c, d])

# Translation of 2 units
T = [
    [1, 0, 2],
    [0, 1, 0],
    [0, 0, 1]
]

color_lut = 'rgbc' #4 colors to represent 4 points

fig = plt.figure()
ax = plt.gca()
xs = []
ys = []
for row in A:
    output_row = T @ row  # @ is matrix multiplication
    x, y, i = output_row
    xs.append(x)
    ys.append(y)
    i = int(i) # convert float to int for indexing
    c = color_lut[i]
    plt.scatter(x, y, color=c)
    plt.text(x + 0.15, y, f"{string.ascii_letters[i]}")
xs.append(xs[0])
ys.append(ys[0])
plt.plot(xs, ys, color="gray", linestyle='dotted')
ax.set_xticks(np.arange(-2.5, 3, 0.5))
ax.set_yticks(np.arange(-2.5, 3, 0.5))
plt.grid()
plt.show()
```



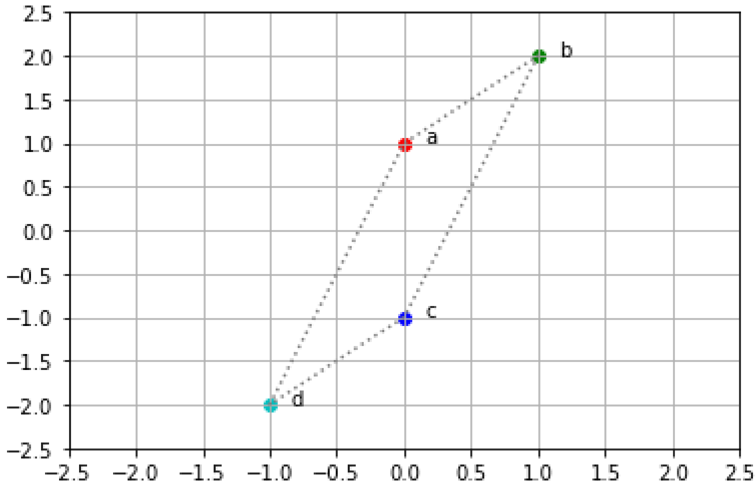
In [6]:

```
# imports for trig
from math import cos, sin, radians

# points a, b and, c
a, b, c, d = (0, 1, 0), (1, 0, 1), (0, -1, 2), (-1, 0, 3)

# matrix with row vectors of points
A = np.array([a, b, c, d])

# Horizontal Shear of 2 units
H = [
    [1, 2, 0],
    [0, 1, 0],
    [0, 0, 1]
]

color_lut = 'rgbc' #4 colors to represent 4 points

fig = plt.figure()
ax = plt.gca()
xs = []
ys = []
for row in A:
    output_row = H @ row  # @ is matrix multiplication
    x, y, i = output_row
    xs.append(x)
    ys.append(y)
    i = int(i) # convert float to int for indexing
    c = color_lut[i]
    plt.scatter(x, y, color=c)
    plt.text(x + 0.15, y, f"{string.ascii_letters[i]}")
xs.append(xs[0])
ys.append(ys[0])
plt.plot(xs, ys, color="gray", linestyle='dotted')
ax.set_xticks(np.arange(-2.5, 3, 0.5))
ax.set_yticks(np.arange(-2.5, 3, 0.5))
plt.grid()
plt.show()
```



In [7]:

```python
# imports for trig
from math import cos, sin, radians

# points a, b and, c
a, b, c, d = (0, 1, 0), (1, 0, 1), (0, -1, 2), (-1, 0, 3)

# matrix with row vectors of points
A = np.array([a, b, c, d])

# Vertical Shear of 2 units
V = [
    [1, 0, 0],
    [2, 1, 0],
    [0, 0, 1]
]

color_lut = 'rgbc' #4 colors to represent 4 points

fig = plt.figure()
ax = plt.gca()
xs = []
ys = []
for row in A:
    output_row = V @ row  # @ is matrix multiplication
    x, y, i = output_row
    xs.append(x)
    ys.append(y)
    i = int(i) # convert float to int for indexing
    c = color_lut[i]
    plt.scatter(x, y, color=c)
    plt.text(x + 0.15, y, f"{string.ascii_letters[i]}")
xs.append(xs[0])
ys.append(ys[0])
plt.plot(xs, ys, color="gray", linestyle='dotted')
ax.set_xticks(np.arange(-2.5, 3, 0.5))
ax.set_yticks(np.arange(-2.5, 3, 0.5))
plt.grid()
plt.show()
```



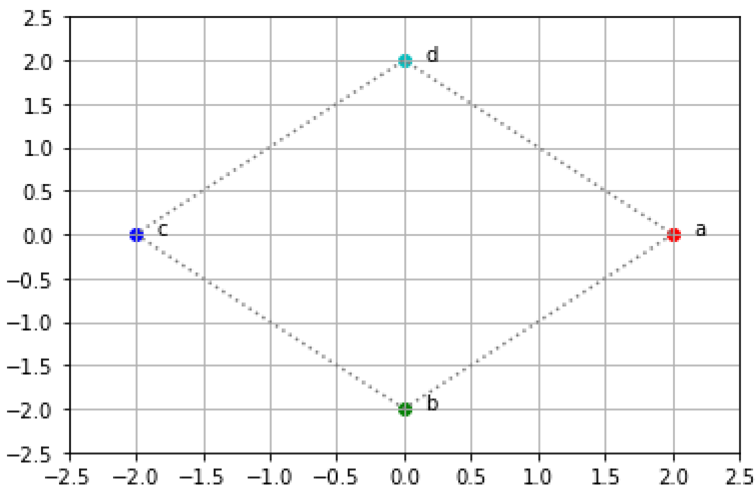
Question 3

Modify the code to implement a combination of the rotation and scaling transformations, i.e., a rotation followed by scaling. Note that since the transformations are linear, a combination of transformations is represented simply as a product of the matrices representing the individual transformation.

In [8]:

```
# imports for trig
from math import cos, sin, radians

# points a, b and, c
a, b, c, d = (0, 1, 0), (1, 0, 1), (0, -1, 2), (-1, 0, 3)

# matrix with row vectors of points
A = np.array([a, b, c, d])

# Rotation Scaling Combo
# rotate by 90deg
# scale by 2

# we have to convert degrees to radians first before we trig them

R = np.array([
    [cos(radians(90)), sin(radians(90)), 0],
    [-sin(radians(90)), cos(radians(90)), 0],
    [0, 0, 1]
])

S = np.array([
    [2, 0, 0],
    [0, 2, 0],
    [0, 0, 1]
])

RS = R @ S

color_lut = 'rgbc' #4 colors to represent 4 points

fig = plt.figure()
ax = plt.gca()
xs = []
ys = []
for row in A:
    output_row = RS @ row  # @ is matrix multiplication
    x, y, i = output_row
    xs.append(x)
```

```
        ys.append(y)
        i = int(i) # convert float to int for indexing
        c = color_lut[i]
        plt.scatter(x, y, color=c)
        plt.text(x + 0.15, y, f"{string.ascii_letters[i]}")
    xs.append(xs[0])
    ys.append(ys[0])
    plt.plot(xs, ys, color="gray", linestyle='dotted')
    ax.set_xticks(np.arange(-2.5, 3, 0.5))
    ax.set_yticks(np.arange(-2.5, 3, 0.5))
    plt.grid()
    plt.show()
```



Exercise 2 - Web Search – PageRank

Question 4 -

In [9]:
```
'''Function to transform a matrix to reduced row echelon form'''
def rref(A):
    tol = 1e-14
    #A = B.copy()
    rows, cols = A.shape
    r = 0
    pivots_pos = []
    row_exchanges = np.arange(rows)
    for c in range(cols):
        ## Find the pivot row:
        pivot = np.argmax (np.abs (A[r:rows,c])) + r
        m = np.abs(A[pivot, c])
        if m <= tol:
        ## Skip column c, making sure the approximately zero terms are
        ## actually zero.
            A[r:rows, c] = np.zeros(rows-r)
        else:
            ## keep track of bound variables
            pivots_pos.append((r,c))

            if pivot != r:
                ## Swap current row and pivot row
                A[[pivot, r], c:cols] = A[[r, pivot], c:cols]
                row_exchanges[[pivot,r]] = row_exchanges[[r,pivot]]

            ## Normalize pivot row
            A[r, c:cols] = A[r, c:cols] / A[r, c];

            ## Eliminate the current column
            v = A[r, c:cols]
```

```
            ## Above (before row r):
            if r > 0:
                ridx_above = np.arange(r)
                A[ridx_above, c:cols] = A[ridx_above, c:cols] - np.outer(v, A[ridx_a
                ## Below (after row r):
            if r < rows-1:
                ridx_below = np.arange(r+1,rows)
                A[ridx_below, c:cols] = A[ridx_below, c:cols] - np.outer(v, A[ridx_b
                r += 1
        ## Check if done
        if r == rows:
            break;
    return A
```

In [10]:

```
from fractions import Fraction as frac

linking_matrix = np.array([
    [frac(0),frac(1,3),frac(1,3),frac(1,2)],
    [frac(1,2),frac(0),frac(1,3),frac(0)],
    [frac(1,2),frac(1,3),frac(0),frac(1,2)],
    [frac(0),frac(1,3),frac(1,3),frac(0)]
    ], dtype=float)

identity = np.eye(4)

A = np.subtract(linking_matrix, identity) #(L-I)r=0 is equivalent to Ax=b; where b=0
# so A = L-I;

S = rref(A)
print(S)
```

```
[[ 1.      0.      0.     -1.5   ]
 [ 0.      1.      0.     -1.3125]
 [ 0.      0.      1.     -1.6875]
 [ 0.      0.      0.      0.    ]]
```

So from the above result we can see that the solution is:

r_a = 1.5

r_b = 1.3125

r_c = 1.6875

r_d = free var

Question 5

In [11]:

```
# define our matrices
linking_matrix = np.array([
    [0  , 1/2, 1/4, 1  , 1/3],
    [1/3, 0  , 1/4, 0  , 0  ],
    [1/3, 1/2, 0  , 0  , 1/3],
    [1/3, 0  , 1/4, 0  , 1/3],
    [0  , 0  , 1/4, 0  , 0  ]
])
identity = np.eye(5)

# rref the diff
A_2 = np.subtract(linking_matrix, identity)
rref(A_2)

A_2
```

Out[11]:

```
array([[ 1.        ,  0.        ,  0.        ,  0.        , -6.33333333],
       [ 0.        ,  1.        ,  0.        ,  0.        , -3.11111111],
```

```
       [ 0.        ,  0.        ,  1.        ,  0.        , -4.        ],
       [ 0.        ,  0.        ,  0.        ,  1.        , -3.44444444],
       [ 0.        ,  0.        ,  0.        ,  0.        ,  0.        ]])
```

Considering the result of the rref, we can deduce that the solution is

r_A = 6.3333
r_B = 3.1111
r_C = 4
r_D = 3.4444
r_E = free var

Question 6

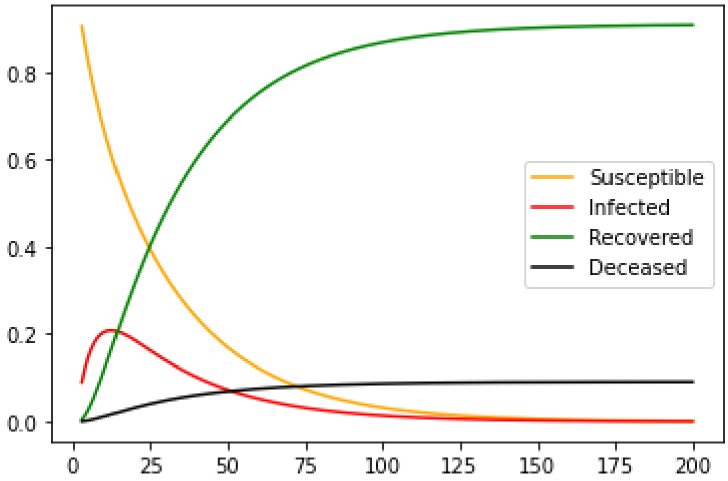
In [12]:
```
x = np.array([
    [0.75], # suseptible
    [0.1], # has the disease currently
    [0.1], # had the disease has recovered and how has immunity
    [0.05]]) # deceased

P = np.array([
    [0.95,0.04,0,0],
    [0.05,0.85,0,0],
    [0,0.10,1,0],
    [0,0.01,0,1]])
x_1 = P @ x
print(x_1)
```

```
[[0.7165]
 [0.1225]
 [0.11  ]
 [0.051 ]]
```

Question 7

In [13]:
```
from numpy.linalg import matrix_power
P = np.array([[0.95 , 0.04 , 0 , 0], [0.05, 0.85, 0,0], [0,0.1,1,0], [0,0.01,0,1]])
x_1 = np.array([1,0,0,0])
S = []
I = []
R = []
D = []
x = []

for i in range(2,200):
    x.append(i+1)
    S.append((matrix_power(P, i) @ x_1)[0])
    I.append((matrix_power(P, i) @ x_1)[1])
    R.append((matrix_power(P, i) @ x_1)[2])
    D.append((matrix_power(P, i) @ x_1)[3])

# plt.subplot(4,1,1)

plt.plot(x, S, color="orange", label = "Susceptible")
plt.plot(x, I, color="red",label = "Infected")
plt.plot(x, R, color="green", label = "Recovered")
plt.plot(x, D, color="black", label = "Deceased")
plt.legend()
```

Out[13]:
```
<matplotlib.legend.Legend at 0x2eb8474ba00>
```


0.8
0.6
0.4
0.2
0.0
0
25
50
75
100
125
150
175
200
Susceptible
Infected
Recovered
Deceased